GYRO Performance on a Variety of MPP Systems

Jeff Candy, General Atomics *and* **Mark Fahey**, ORNL

ABSTRACT: GYRO is a code used for the direct numerical simulation of plasma microturbulence. It is the most physically comprehensive of all such codes, worldwide. It has been ported to a variety of modern MPP platforms including a number of commodity clusters, IBM SPs and the Cray X1. The performance and scaling of GYRO on many of these systems has been previously documented, and exhaustive testing and performance analysis has already been the subject of numerous presentations and publications. Although we briefly describe the mathematical structure of the equations solved by GYRO, we prefer to focus on the data layout and redistribution algorithms. Performance comparisons of different platforms are detailed, with emphasis on a section of code which requires very large communication bandwidth. In addition, we discuss the performance of a recently implemented FFT algorithm for the evaluation of the nonlinear terms. This algorithm gives a performance improvement for very large grid sizes. However, because the FFT vector length very rarely exceeds 192 in practice, one never operates the code in a regime where the full aymptotic dominance of the FFT over the original direct method is realized.

KEYWORDS: gyrokinetic, plasma, fusion, AMD, Infiniband, IBM, X1, FFT.

1 Introduction

The most promising and aggressively studied concept for power production by fusion reactions is the tokamak. At the present time, magnetic fusion energy research has reached the point where construction of a tokamak burning plasma facility such as the proposed ITER experiment [1] is prudent. The design of a next-step device like ITER calls for thermonuclear heating to balance the transport and radiation losses for periods of 10^3 seconds or more. However, despite the advances made in the understanding and control of tokamak plasmas, some uncertainties remain in reliably predicting confinement properties and performance of next-generation ITER-scale devices. Within the worldwide fusion community, it is widely agreed that the so-called gyrokinetic-Maxwell (GKM) equations [2, 3] provide a solid foundation for the first-principles calculation of turbulent tokamak heat and particle transport. For years, the numerical solution of the nonlinear GKM equations has been a computational physics "Grand Challenge". Development of GYRO was partially funded by the *Plasma Microturbulence Project*, a fusion SciDAC project.

The code, written primarily in Fortran 90, solves the nonlinear GKM equations for core ions, electrons and any number of plasma impurity species. Electromagnetic or reduced electrostatic simulations are possible. The computational domain can be radially global, with input data taken from an experimental database, or radially local, in order to accurately study isolated parametric effects. Both partial and full torus simulations are possible, although the latter is rarely (if ever) necessary to obtain a converged result. GYRO uses a five-dimensional Eulerian grid and advances the system in time using a second-order, implicit-explicit (IMEX) Runge-Kutta (RK) integrator [4]. There is also an option to use a fourth-order explicit Runge-Kutta time advance when one of the species responds adiabatically (an approximation). In recent years, the Eulerian approach has proven to be more successful than straightforward Lagrangian (or particle-in-cell) discretizations, in particular because the particle noise inherent to the latter makes long-time simulation virtually impossible using a reasonable number of particles. This issue is presently the subject of great controversy.

GYRO has been ported to a variety of modern MPP platforms including a number of commodity clusters, IBM SPs and the Cray X1. It has shown good scalability on all these platforms. The GYRO users group (comprised of both students and researchers) continues to evolve. With this evolution comes ever greater expectations for code documentation, robustness and ease-of-use.

The performance and scaling of GYRO on MPP systems has been previously documented [5], and is the subject of ongoing research. In the present report, we focus on

1. the multidimensional grid layout, and performance of the associated data redistribution algorithms; and
2. the performance of an alternative scheme (FFT-based) for evaluation of the nonlinear convolution (Poisson bracket).

The remainder of the paper is organized as follows. Section 2 gives a heuristic overview of the gyrokinetic model [2], describes the connection of the physical variables to gridpoints, and outlines the basic grid distribution scheme. Section 3 discusses numerical results, including various performance measurements. Section 4 lists the essential code used in the GYRO communication calls. Finally, a brief summary is given.

2 Gyrokinetic model overview

2.1 Form of the equations

The GKM equations couple the *gyrocenter* distribution, f, to the electromagnetic fields, Φ:

$$\frac{\partial f}{\partial t} = \mathcal{L}_a f + \mathcal{L}_b \langle \Phi \rangle + \{ f, \langle \Phi \rangle \} \tag{1}$$

$$\mathcal{F}\Phi = \int\int dv_1\, dv_2\, \langle f \rangle \tag{2}$$

$\mathcal{L}_b$, $\mathcal{L}_b$ and $\mathcal{F}$ are linear operators, and $\langle \cdot \rangle$ is an operator which takes the average along a particle gyro-orbit. Strictly speaking, f measures the deviation from a Maxwellian background. For global simulations, a linear adaptive source technique [6] is used to inhibit the evolution of f and Φ on equilibrium scales. The sole nonlinearity, which has a Poisson bracket structure, appears in the gyrokinetic equation. The function $f(\mathbf{r}, v_1, v_2)$ is discretized over a 5-dimensional grid (three spatial and two velocity coordinates), while the 3-dimensional electromagnetic fields $\Phi(\mathbf{r}) = [\phi, A_\parallel]$ are independent of velocity. Here ϕ and $A_\parallel$ are the *electrostatic* and *electromagnetic* potentials, respectively. In order to obtain Eq. (1), one has averaged over the fast orbital motion (gyro-orbit) to eliminate the third velocity-space dimension (gyro-angle). However, this so-called *gyro-averaging* operation introduces *nonlocal* spatial operators, $\mathcal{F}$ and $\langle \cdot \rangle$, perpendicular to the magnetic field.

We remark that simulations normally reach a statistical steady state on a timescale much shorter than an energy confinement time. In practice, simulations are well into the steady-state regime after about 50,000 timesteps.

2.2 Grid indexing

Eulerian schemes for solving the GKM equations evolve the gyro-center distribution function $f(r, \tau, n_{\mathrm{tor}}, \lambda, E)$, where r is the plasma minor radius, τ is the orbit time (a parametrization of the poloidal angle), n_{tor} is the toroidal mode number (a linear quantum number), λ is the cosine of the pitch angle and E is the energy. Upon discretization of all differential and integral operators, we solve difference equations for the quantities $f(i, j, n, k, e)$, with

$$i = 1, 2, \ldots, N_i \tag{3}$$
$$j = 1, 2, \ldots, N_j \tag{4}$$
$$n = 1, 2, \ldots, N_n \tag{5}$$
$$k = 1, 2, \ldots, N_k \tag{6}$$
$$e = 1, 2, \ldots, N_e \tag{7}$$

Note that in general there is also a species index, but in the present work we do not use it for data distribution.

2.3 Grid distribution and redistribution

While current supercomputers offer the promise of multi-Tflop/s performance, the newest and most powerful of these are based on a distributed-memory architecture and require efficient data distribution schemes to achieve good parallel performance. The prototype object to be distributed in memory evenly across the entire processor space is the function $f(i, j, n, k, e)$. The distribution of an index across processors is incompatible, however, with the evaluation of operators on that index. For example, a derivative in r requires that all i should be on a processor. Even more complicated is the requirement that some operators require that more than one index be simultaneously on-processor. For example, the nonlinear convolution requires both i and n to be on-processor. A summary of distribution requirements is given in Table 1.

Stage	**On-processor indices**
Linear with field solve	i, j
Pitch-angle scattering	j, k
Energy diffusion	e
Nonlinear	i, n

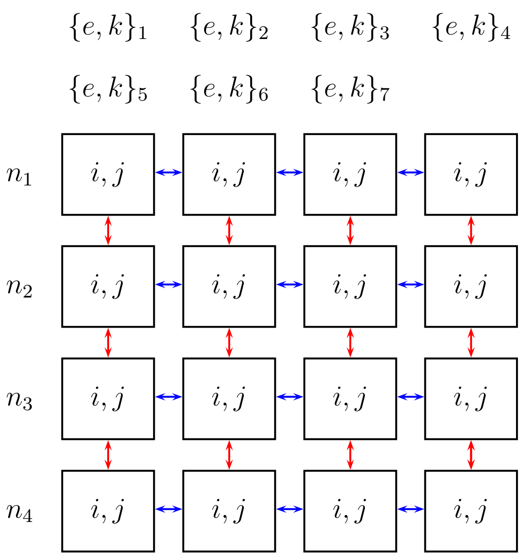
Table 1: Distribution requirements for different code stages (i.e., evaluation of different operators).

The obvious tactic is to change the distribution scheme to evaluate different operators. We will describe a algorithm with uses not only the global communicator `MPI_COMM_WORLD` for the totality of processors, but two new communicators, `COMM1` and `COMM2` which define processor rows and columns.

2.3.1 Base distribution scheme

The base distribution scheme is that used for the *linear step with field solve*

$$\textbf{BASE:} \qquad f([n], \{e,k\}, i, j) \tag{8}$$



$\{e,k\}_1$ $\{e,k\}_2$ $\{e,k\}_3$ $\{e,k\}_4$
$\{e,k\}_5$ $\{e,k\}_6$ $\{e,k\}_7$
n_1
n_2
n_3
n_4
i,j

Figure 1: Base grid distribution scheme, with all i, j on processor, n distributed along rows, and e, k distributed along columns. Data redistribution will occur only only along rows or columns, thus limiting the all-to-all exchange size.

Fig. 1 illustrates this distribution strategy in the case of 16 processors – with 4 processors in each of the 4 `COMM1` subgroups, and 4 processors in each of the 4 `COMM2` subgroups. Here, `COMM1` links all columns of a given row (horizontal arrows), and `COMM2` links rows of a given column (vertical arrows). The curly braces indicate a one-dimensional array of *stacked* indices

$$\{e,k\}_p = p \quad \text{with} \quad p = 1, 2, \ldots, N_e\, N_k \tag{9}$$

where

$$(e-1) = \frac{p-1}{N_k} \quad \text{and} \quad k-1 = (p-1) \mod N_k \tag{10}$$

The indices $\{e,k\}$ are distributed along processor columns, the $[n]$ are distributed along rows, and i and j are stored on-processor. Note that in the general case, many stacked $\{e,k\}$ indices will appear in each column. For the 16-processor case shown in Fig. 1, we would have $\{e,k\}_p$ for $p = 1, 5, 9, \ldots$ in column 1, and so on. With regard to columns, however, we make an important **simplifying assumption** and consider the number of rows to be exactly equal to N_n. This restriction suits our immediate purposes well enough, and generalization to more than one n per row is reasonably straightforward.

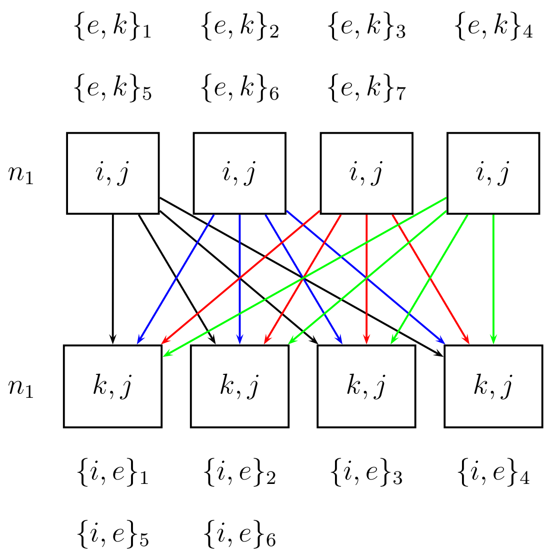
2.3.2 3-index row transpose

We can define a generalized 3-index transpose operator, R, which acts individually on processor rows

$$R : \{e,k\}, i \longrightarrow \{i,e\}, k \tag{11}$$

The omitted index, j, is left on-processor. This operation is illustrated schematically in Fig. 2. Because there are three indices, three applications of the operator R yields the identity:

$$\begin{aligned} R^3 : \{e,k\}, i &= R^2 : \{i,e\}, k && (12)\\ &= R : \{k,i\}, e && (13)\\ &= \{e,k\}, i && (14) \end{aligned}$$



$\{e,k\}_1$ $\{e,k\}_2$ $\{e,k\}_3$ $\{e,k\}_4$
$\{e,k\}_5$ $\{e,k\}_6$ $\{e,k\}_7$
n_1
i,j
n_1
k,j
$\{i,e\}_1$ $\{i,e\}_2$ $\{i,e\}_3$ $\{i,e\}_4$
$\{i,e\}_5$ $\{i,e\}_6$

Figure 2: Schematic description of a row exchange operation. Before the exchange, all i, j are on-processor. After the exchange, all k, j are on-processor.

Pointers marking the location of data after a transpose are computed and stored in a transpose initialization routine. A formal study of the mathematical properties of this transpose operation might prove enlightening. Indeed, the creation of standard libraries for the permutation of array indices as described in the report would be of enormous benefit for the plasma microturbulence community.

2.3.3 2-index column transpose

Figure 3 shows the transpose operation on COMM2 required to place all values of n on-processor and distribute the j index. This exchange must accomodate $N_j \geq N_n$. So, generally, in row i we would have $j_i, j_{i+4}, \ldots$. Note, however, that load balancing will be less than optimal unless N_j is an integer multiple of N_n. A particularly inefficient case is $N_n > N_j$ (many toroidal modes with low poloidal resolution), which fortunately does not occur often in practice. Production simulation almost exclusively use $N_n = (8, 12, 16)$ and $N_j = (20, 28, 32)$.

In the mathematical sense, the $n \leftrightarrow j$ exchange is a special case of the COMM1 transpose. For greater efficiency, we implement this 2-index transpose separately.

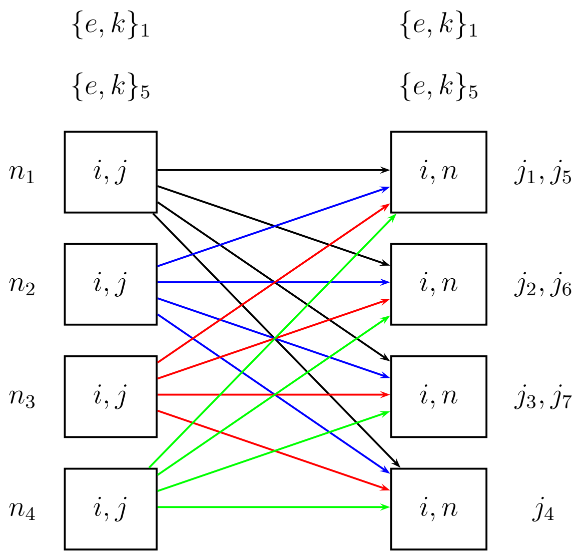


Figure 3: Schematic description of a column exchange operation. Before the exchange, all i, j are on-processor. After the exchange, all i, n are on-processor.

2.3.4 Index distribution summary

A summary of the f-distribution in each stage in given in Table 2.

Stage	**Distribution**
Linear with field solve	$f([n], \{e,k\}, i, j)$
Pitch-angle scattering	$f([n], \{i,e\}, k, j)$
Energy diffusion	$f([n], \{k,i\}, e, j)$
Nonlinear	$f([j], \{e,k\}, i, n)$

Table 2: Distributions of f in each stage.

A partial code listing for the row and column transpose operations is given in the Section 4.

2.4 Discretization schemes

We briefly sketch the type of discretization scheme used in each dimension. A detailed treatment is beyond the scope of the present paper.

- $r \to i$ (radius): linear advective derivatives on f are treated with an upwind differences, whereas derivatives on fields are treated with centered differences. The nonlocal operators $\mathcal{F}$ and $\langle \cdot \rangle$ are approximated using a (banded) pseudospectral technique. The order of all discretizations is adjustable at run-time.

- $\tau \to j$ (poloidal angle): for f, there is no fixed grid in poloidal angle, θ. Instead, the transformation
$$\frac{v_\parallel(r,\lambda,\theta)}{R_0 q(r)} \frac{\partial}{\partial \theta} \to \Omega(r,\lambda) \frac{\partial}{\partial \tau} \qquad (15)$$
is used to eliminate the singularity at bounce points, $v_\parallel(\theta) = 0$. Here, $v_\parallel$ is the velocity parallel along the magnetic field, R_0 is the major radius of the torus, and q is the so-called *safety factor*. Then, an upwind scheme in τ is used to discretize $\partial f / \partial \tau$. The use of a τ-grid (leading to a different set of points in θ for every value of λ) for the GK equation dictates that the Maxwell equations are solved by expansion of fields in complex finite-elements: $\phi(r_i, \theta) = \sum_m F_m^i(\theta)$. The F_m^i satisfy the a complex phase condition.

- $n_{\text{tor}} \to n$ (toroidal angle): the toroidal direction (really, the direction perpendicular to both the radius, r, and to the magnetic field) is treated in a fully spectral manner. Note that simulations need not cover an entire toroidal circuit $(0, 2\pi]$. In fact, it is normally most efficient to cover a partial torus; for example: $n_{\text{tor}} = 0, 10, 20, \ldots$.

- $(\lambda, \epsilon) \to (k, e)$ (velocity-space): A transformation property under integration of velocity-space integrals over θ is used to recast the velocity-space integration. Then, in both ϵ and λ, an exact Gauss-Legendre quadrature scheme is numerically generated (by nonlinear root-finding) at run-time. This is different at each radius and for different plasma equilibria.

- **nonlinearity:** The nonlinear Poisson bracket is evaluated with a *conservative* difference-spectral analogue of the Arakawa method. This scheme ensures exact conservation of density and generalized entropy at vanishing time step (independent of grid resolution).
- **collisions:** Collisions are represented by a second-order diffusive-type operator in λ. This operator is split from the collisionless problem and a irregular-grid generalization of the Crank-Nicholson method is used.
- **time-advance:** A 2nd-order IMEX RK scheme is used, with the electron parallel motion ($\partial/\partial\theta$) treated implicitly. This is exceptionally complicated due to the use of a τ-grid, as well as the presence of the fields in the advection. However, the implicitness is crucial for the elimination of a numerical instability connected with pathological *electrostatic Alfvén waves.*

3 GYRO Performance

In this section we describe the performance of GYRO on five separate platforms. As mentioned previously, GYRO has been ported to a variety of modern MPP systems including a number of commodity clusters, IBM SPs, an SGI Altix, and the ORNL Cray X1. Since the developers desire portable code rather highly, only a single source is maintained. In a few exceptional cases, there are platform-specific routines (FFT, MPI-IO). In general, a port to a new architecture requires little more than the creation of a new makefile. This is mentioned now because although the single-source philosophy has made GYRO a very portable code, the single-source requirement has the consequence of preventing some machine-specific code optimizations which adversely affect performance on other architectures. However, this requirement does not prevent machine specific optimizations altogether. For example, GYRO was modified (making code vector friendly, adding directives) to eliminate bottlenecks on the X1. However, these changes were minimal and in some cases simultaneously beneficial on other architectures. In porting to the X1, the code modifications were largely the addition of directives, but there were selected instances of rank promotion/demotion, and an instance of "pushing" a loop down into a subroutine call.

We now summarize the hardware features of the five platforms used for GYRO performance testing. The reader should note that some of this data has been previously reported [5].

1. **AMD Opteron-IB cluster**
The General Atomics AMD Opteron cluster with Infiniband interconnect has 57 2-way Opteron 250 (2.4 Ghz) nodes running in 64-bit mode. The peak performance of a single processor is $2.4 \times 2 = 4.8$ GFlops/s. GYRO was compiled using version 6.0 of pgf90. The machine uses the mvapich 0.9.5 implementation of MPI over Infiniband. The network topology is that of three subclusters, each serviced by a 24-port Infiniband switches. These switches are **not connected** to one another. For this reason, the maximum job size is 48 processors on subcluster 1, 48 on subcluster 2 and 18 on subcluster 3. The entire cluster is serviced by a single NFS filesystem over gigabit ethernet.

2. **IBM Nighthawk II (Power3) cluster**
The IBM Nighthawk II cluster with Colony interconnect has 416 16-way Power3 (375 Mhz) nodes. The peak performance of a Power3 processor is $375 \times 4 = 1.5$ GFlops/s. GYRO was compiled using ESSL 3.3 and XL Fortran 8.1. The machine has parallel environment 3.2 and AIX 5.1. It is managed by NERSC and represents the workhorse platform for the US fusion program.

3. **IBM p690 (Power4) cluster**
The IBM p690 cluster with Federation interconnect has 27 32-way Power4 (1.3 GHz) nodes. The peak rate of each node is $32 \times 1.3 \times 4 = 166.4$ GFlops/s. GYRO was built using ESSL 4.1 and XL Fortran 8.1. The machine, managed by ORNL CCS, has parallel environment 4.1 and AIX 5.2.

4. **SGI Altix**
The SGI Altix is a single-system image running Linux with 256 Itanium 2 processors (1.5 GHz). The machine peak rate is $256 \times 1.5 \times 4 = 1536$ GFlops/s. GYRO was built using the Intel Fortran 8.0 compiler and SGI's SCSL library. This system is managed by ORNL CCS.

5. **Cray X1**
The Cray X1 is also a single-system image of 504 multistreaming processors. Each processor is capable of 12.8 Gflops/s, and thus the peak of the machine is 6451.2 Gflops/s. GYRO was built using Programming Environment 5.2.0.2 and mpt 2.3.0.4. The OS was Unicos m/p 2.4.17. This system is managed by ORNL CCS.

3.1 Waltz Standard Case benchmark

For an overall system comparison, we consider results from a benchmark based on the *Waltz Standard Case* parameters [7, 8]. This case, which is meant to be representative of the core plasma in a medium-sized tokamak, includes kinetic electron dynamics and collisions. The numerical grid resolution used for this benchmark is typical of that used in production runs. It represents, roughly, the *minimum grid size* required to obtain physically accurate results. We could have improved the scalability on all platforms by moving to a finer grid, but felt that results for a grid of typical production size are of greater practical value. This benchmark exercises most of the code capability, including the pitch-angle collision operator, for which there have been historical performance problems on the X1.

Figure 4 shows the results of running the aforementioned benchmark on each platform.

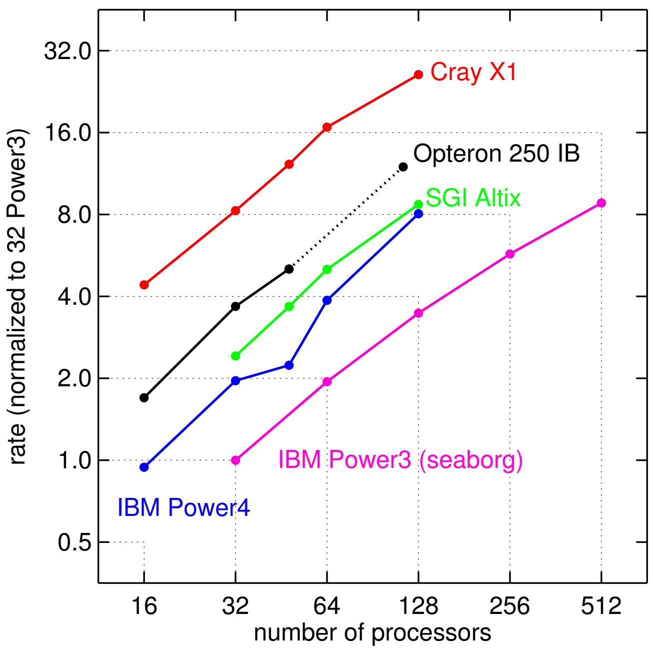


Figure 4: Comparison of overall performance on fixed-problem-size simulations of the *Waltz Standard Case*. The vertical axis measures the *execution rate*, normalized to the execution rate on 32 Power3 processors. The scaling is very good on all platforms, with the X1 a clear winner in per-processor performance. The dotted line on the Opteron-IB curve represents the *expected scaling* if a single switch were to cover the whole cluster.

While some of the data in Fig. 4 has been presented previously [5], we have updated the plot to show results for the recently-installed General Atomics Opteron cluster.

3.2 Column-transpose performance

Next, we run a simplified benchmark case in order to assess the performance of the column transpose (required for the Poisson bracket evaluation) on three of the previous platforms (IBM, Cray and Opteron-IB) for a **fixed** problem size. The results, shown in Fig. 5, illustrate the excellent scalability with processor count achieved on the Cray and Opteron-IB systems. The scalability of the IBM Power3 system on this test is somewhat less impressive. With regard to actual communication time (rather than scalability) the Cray X1 shows itself to be nearly a factor of 10 faster than the Opteron-Infiniband system. The interconnect performance of the Cray X1 is truly impressive.

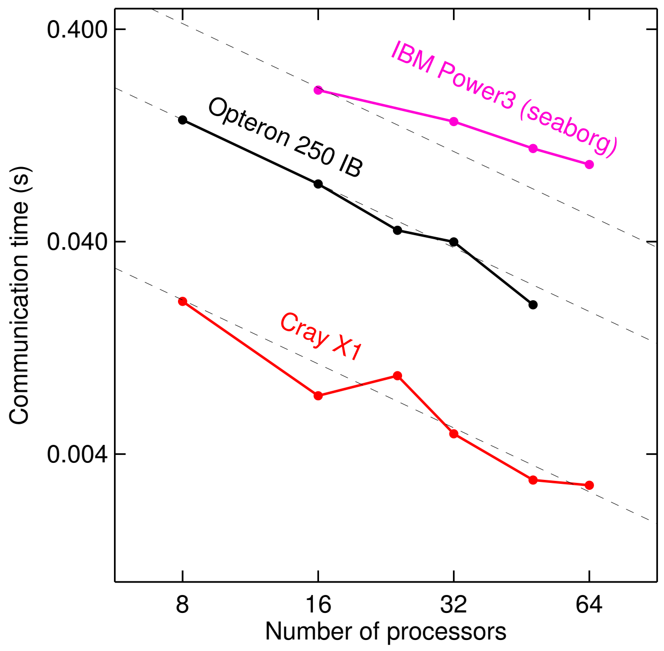


Figure 5: Absolute communication time for forward plus reverse column transpose. The Cray X1 is by far the best performer, with the Opteron-IB (infiniband) system a distant second. The IBM Power3 comes in last, with marginal network scalability and lowest total throughput. Dotted lines show the theoretical $1/n_{\text{proc}}$ trend, normalized to unity at 8-processors.

At these modest processor counts (the range 32 to 64 is representative of real production simulations), where lack of scalabilty is not yet a show-stopper, it is interesting to compare the *relative fraction of time spent in communication*. Figure 6 shows the communication-to-computation ratio for the same scenario as considered in Fig. 5. The salient point is that the AMD and IBM are about equally well-balanced, both experiencing a substantial communication bottleneck. The X1 is the only platform for which the normally bandwidth-hungry GYRO code does not see a significant communication overhead.

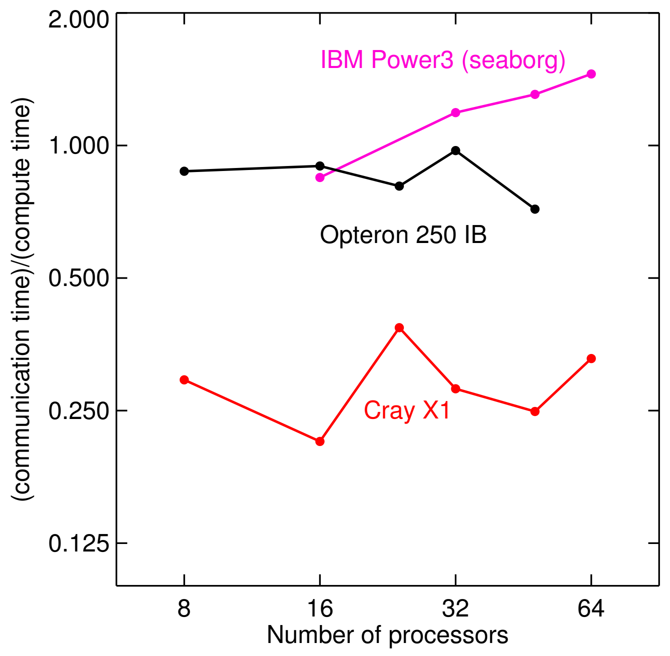


Figure 6: Ratio of communication time to computation time in the evaluation of the nonlinear term in Eq. (1). The ratio here is a measure of the communication overhead on a given platform. By this measure, the communication-to-computation ratios are equally well-balanced on the IBM and AMD systems. In some sense, the results shows that the Cray interconnect is over-engineered!

3.3 Direct vs. FFT methods

On these same three platforms, we also wish to compare the relative performance of the direct and FFT methods for evaluating the Poisson bracket nonlinearity. Both methods yield, in principle, exactly the same result for the bracket. In practice, there is a small difference due to round-off error. Each system uses a different FFT library: the AMD uses FFTW 2.1.5, the IBM uses ESSL, and the Cray uses LibSci.

Fig. 7 compares the direct and FFT methods for an **increasing** problem size. Here, we have set the number of complex modes equal to the number of processors. Because of dealiasing, the FFT length in each case is $3N_n$ (three times the number of modes). Because the number of processors used must be an integer multiple of N_n, we use N_n processors for N_n modes.

The results have a relatively straightforward interpretation: the direct method (solid curve) is preferred over the FFT method (dotted curve) at

- 16 modes on the IBM,
- 32 modes on the AMD,
- 48 modes on the X1.

One unfortunate feature of this test is that the poloidal grid dimension is rather small: $N_j = 20$. Therefore, as discussed in Section 2, the distribution scheme becomes poorly load balanced for $N_n > 20$. We can remove this artifact by choosing $N_j = 64$ (and halving N_e) and rerunning the tests. We also plot only $N_n = (16, 32, 64)$ so that N_j is evenly divisible by N_n. This rectified data is shown in Fig. 8 for the Cray and IBM.

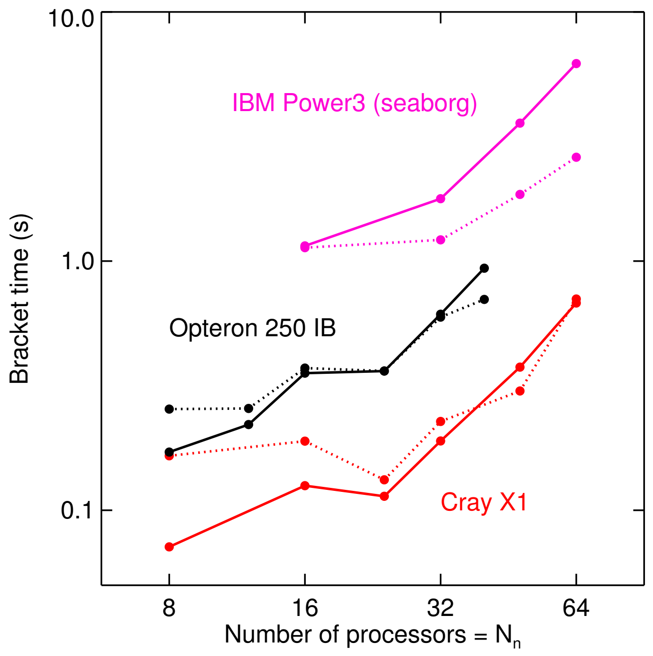


Figure 7: Comparison of direct and FFT schemes for the nonlinear bracket (convolution) for problem size which **grows** in proportion to the number of Fourier modes, N_n. The FFT length is three times the number of processors (or three time the number of modes). The direct method (solid curves) is preferred over the FFT method (dotted curves) at 16 modes on the IBM, 32 modes on the AMD, and 48 modes on the X1. Here, $N_j = 20$.

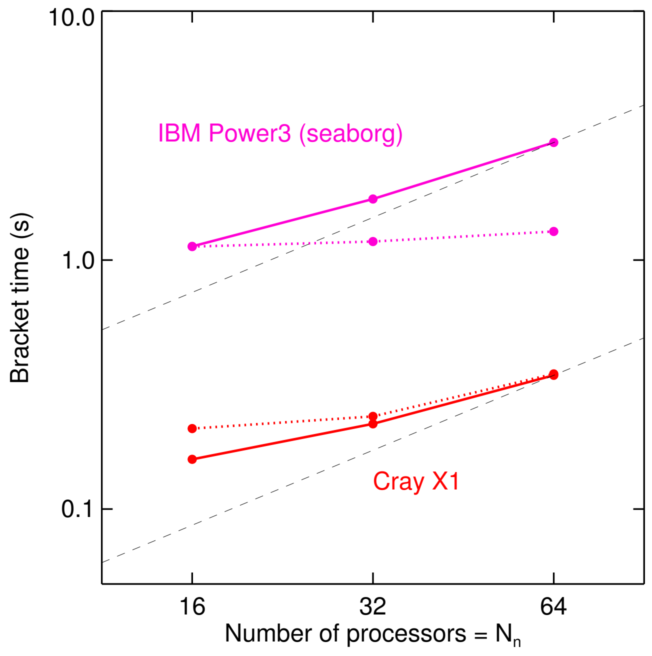


Figure 8: Comparison of direct and FFT schemes for the nonlinear bracket (convolution) for problem size which **grows** in proportion to the number of Fourier modes. This case has $N_j = 64$ to ensure uniform load-balancing. The dashed lines indicate the scaling for an N_n^2 operation count (divided into N_n processors).

The dashed lines in the figure indicate the theoretical N_n-scaling, which reflects an overall N_n^2 scaling of the direct method. Evidently, for typical grid sizes, the FFT length is never long enough to achieve the asymptotic $N_n \log N_n$ scaling of the FFT. How these results change with increasing radial grid size (the radial grid was relatively small for this test case) needs further study. Although at present simulations are rarely large enough to warrant the FFT method, we expect that as simulations grow in size and complexity, its usage will become more frequent.

4 Code listing

4.1 Row Transpose

```
subroutine fTRANSP_DO(g,gT)

  use fTRANSP_GLOBALS

  implicit none

  complex, intent(in), &
   dimension(n_ij_loc,n_k) :: g
  complex, intent(inout), &
   dimension(n_jk_loc,n_i) :: gT

  include 'mpif.h'

  ! Sort g into packages to be sent to each
  ! processor: q_send(:,i_recv)

  s = 0
  p_ij_loc = 0
  do p_ij=1+i_proc,n_ij,n_proc

     j = j_ij(p_ij)
     p_ij_loc = p_ij_loc+1

       do k=1,n_k

          p_jk      = jk(j,k)
          i_recv    = modulo(p_jk-1,n_proc)
          s(i_recv) = s(i_recv)+1

          q_send(s(i_recv),i_recv) = g(p_ij_loc,k)

     enddo ! k

  enddo ! p_ij

  ! Do all-to-all exchange of q_send
  ! into q_recv

  call MPI_ALLTOALL(q_send, &
                    s_dim, &
                    MPI_DOUBLE_COMPLEX, &
                    q_recv, &
                    s_dim, &
                    MPI_DOUBLE_COMPLEX, &
                    TRANSP_COMM, &
                    i_err)

  ! Build "transposed" g (gT) from packages
  ! sent by all other processors.

  do i_from=0,n_proc-1
     do s0=1,s_dim
        i = i_map(s0,i_from)
        p_jk_loc = p_jk_loc_map(s0,i_from)
        if (i > 0 .and. p_jk_loc > 0) then
           gT(p_jk_loc,i) = q_recv(s0,i_from)
        endif
     enddo
  enddo

end subroutine fTRANSP_DO
```

4.2 Column Transpose

```
subroutine fSSUB(x_IN,xt)

  use SSUB_private

  !------------------------------------------
  implicit none
  !
  complex, intent(in), &
           dimension(nv1,nv2,nj) :: x_IN
  complex, dimension(nv1,nv2,nn*jsplit) :: x
  complex, intent(inout), &
           dimension(nv1,nv2,jsplit,nn) :: xt
  !------------------------------------------

  include 'mpif.h'

  do j=1,nj
     x(:,:,j) = x_IN(:,:,j)
  enddo
  do j=nj+1,nn*jsplit
     x(:,:,j) = (0.0,0.0)
  enddo

  do j=1,jsplit

     j1 = 1+(j-1)*nn
     j2 = j1+nn-1

     call MPI_ALLTOALL(x(:,:,j1:j2), &
          nv1*nv2, &
          MPI_DOUBLE_COMPLEX, &
          xt(:,:,j,:), &
          nv1*nv2, &
          MPI_DOUBLE_COMPLEX, &
          SSUB_COMM, &
          i_err)

  enddo

end subroutine fSSUB
```

Summary

We have outlined the form of the equations used for massively parallel gyrokinetic simulations, and an array distribution/redistribution scheme appropriate for the Eulerian solution of said equations. An analysis of the performance of these *generalized transpose* routines shows good communication scalability at fixed problem size on the slowest (IBM Power3) and fastest (Cray X1) systems. Because data is normally (but not always) distributed in such a way as to maintain perfect load balancing, the overall code scalability is very good. We have also made a first attempt to measure the relative performance of an alternative FFT scheme for evaluating the nonlinear terms in the GKM equations. Rough indications of the crossover point on each computer as a function of N_n have been given.

Acknowledgements

This work was supported by U.S. Department of Energy Grant DE-FG03-95ER54309, and also by the Office of Mathematical, Information, and Computational Sciences Division, Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

About the Authors

Jeff Candy is a Principal Scientist in the Energy Group at General Atomics. Jeff obtained his Ph.D. in physics from UCSD under the supervision of the late M.N. Rosenbluth. He can be reached at General Atomics, P.O. Box 85608, San Diego, CA 92186-5608, E-Mail: candy@fusion.gat.com.

Mark R. Fahey is a senior Scientific Application Analyst in the Center for Computational Sciences (CCS) at Oak Ridge National Laboratory. He is the current CUG X1-Users SIG chair. Mark has a Ph.D. in mathematics from the University of Kentucky. He can be reached at Oak Ridge National Laboratory, P.O. Box 2008 MS6008, Oak Ridge, TN 37831-6008, E-Mail: faheymr@ornl.gov.

References

[1] ITER Physics Basis Editors. *Nucl. Fusion* **39**, 2175 (1999).

[2] T. Antonsen and B. Lane. *Phys. Fluids* **23**, 1205 (1980).

[3] M. Kotschenreuther, G. Rewoldt, and W.M. Tang. *Comput. Phys. Commun.* **88**, 128 (1995).

[4] L. Pareschi and G. Russo. in *Hyperbolic problems: Theory, Numerics, Applications.* Springer, (2002).

[5] M.R. Fahey and J. Candy. in *ACM/IEEE SC2004* (, Pittsburgh, PA, 2004).

[6] R.E. Waltz, J. Candy, and M.N. Rosenbluth. *Phys. Plasmas* **9**, 1938 (2002).

[7] R.E. Waltz, G.R. Kerbel, and J. Milovich. *Phys. Plasmas* **1**, 2229 (1994).

[8] J. Candy and R.E. Waltz. *J. Comput. Phys.* **186**, 545 (2003).